Bringing the Multicore Revolution to Safety-Critical Cyber-Physical Systems*

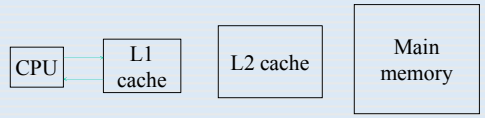
[1]University of North Carolina Chapel Hill [2]North Carolina State University
PIs: Dr. James Anderson[1] & Dr. Frank Mueller[2]
Students: Bryan Ward[1], Jonathan Herman[1], Christopher Kenna[1], Shrinivas Panchamukhi[2]
*Funded in part by NSF awards 1239135 and 1239246.

Motivation

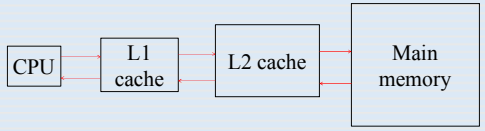
➢Shared hardware like caches & TLBs introduce timing unpredictability for real-time systems (RTS).

➢Worst-case execution time (WCET) analysis for RTS with shared hardware resources often so pessimistic that extra processing capacity of multicore systems is negated.

Problem



➢a) Memory ref hits in L1 cache – Access latency: 1-4 cycles.



➢b) Memory ref misses L1 & L2 cache – Access latency: 40-100 cycles.
➢c) Memory ref misses in TLB – Access latency: +1000 cycles.

➢Tighter WCET estimates can be established if we know which references hit in the cache and which do not.

➢Other shared resources like TLBs show similar timing unpredictability.

Solution

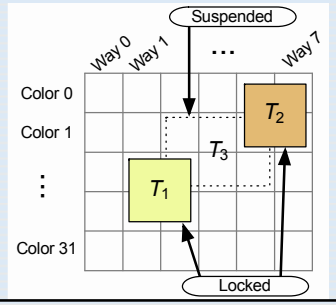
➢ Our solutions focus on two shared resources: shared caches and TLBs.
➢ **Cache Locking:**
- Apply a multiprocessor real-time locking protocol to cache colors.
- Treat each job as a critical section.

➢ **Cache Scheduling:**
- Apply existing scheduling algorithms (e.g., Rate Monotonic) to cache accesses.
- Allows for preemptions w.r.t. the cache (see example).

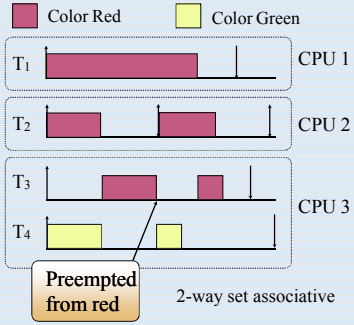
➢ **Reverse engineer the working of TLBs**
- Perform experiments to validate our understanding of the TLBs on different architectures.
- Gain knowledge on the architectural advances made to TLBs.

Caches – Solutions & Results

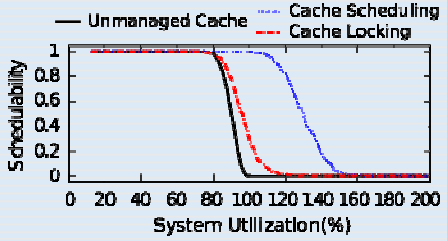
Cache Locking



Cache Scheduling



Overhead-Aware Schedulability Results



Cache locking and cache scheduling, significantly improve hard real-time schedulability.

TLBs – Solution & Results

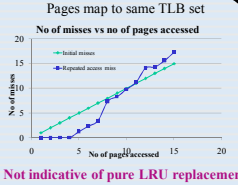
Pseudo code for TLB reverse engineering

```
1:……
2: . . . . .
3: //allocate a huge array
4: int * data = (int *)
      calloc(numOfElements,4);
5: pageOffset = 0;
6:
7: //access pages for the first time
8: PAPI read(eventSet, value1);
9: for i = 0 to noOfPagesToAccess do
10: data[pageOffset] = 1;
11: pageOffset = pageOffset + ( s *
1024);
12: end for
13: PAPI read(eventSet, value2);
14: initial misses = value2 - value1;
15:
16: //access pages repeatedly
17: GetTimeStamp() nnread TSC
register
18: for i = 0!n do
19: pageOffset = 0;
20: PAPI read(eventSet, value3);
21: for j = 0!noOfPagesToAccess
do
22: temp = data[pageOffSet];
23: process(temp);
24: pageOffset = pageOffset + (s*
1024);
25: end for
26: PAPI read(eventSet, value4);
DTLB misses = DTLB misses +
(value4 -value3);
28: calculateMaxMisses()
29: calculateMinMisses()
30: end for
31: GetTimeStamp()
```

Consecutive pages

High no. of misses past the TLB size for x86 Xeon E5

Pages map to same TLB set



Not indicative of pure LRU replacement.

Multiple runs with no of pages =1

Max bounds not deterministic.

Pages map to same TLB set, no of Repeated accesses = 1

Page accesses and cycles not proportional.

Conclusions

➢ Developed 2 techniques based on cache coloring
→ eliminate cross-core cache evictions.
➢ Implemented in a mixed-criticality scheduler: LITMUS^{RT}
➢ Evaluated on an ARM Tegra 3 platform
➢ Conducted overhead-aware schedulability study
→ based on measured overheads.
➢ Cache scheduling & cache locking → improved schedulability
→ over a system with unmanaged cache.
➢ TLBs not using LRU replacement → maybe PLRU (ongoing work)
➢ TLB-miss bounds not deterministic → even for accessing <4 pages.
➢ Current work: mechanisms & policies for TLB predictability